AN ADAPTIVE SOFTWARE RELIABILITY PREDICTION APPROACH

Meng-Lai Yin[*] *Lawrence E. James* *Samuel Keene* *Rafael R. Arellano* *Jon Peterson*
Raytheon Systems Company
Loc. FU. Bldg. 675, M/S AA341
1801 Hughes Drive, Fullerton, CA 92834 USA

ABSTRACT

Software reliability analysis is inevitable for modern systems, since a large amount of system functionality is now dependent on software, and software does contribute to system failures. Although extensive research efforts have been devoted to the field of software reliability, there is no single consensus model available. On the other hand, most software reliability models are based on software failure data collected from the project. This creates a problem for the designers since, during the early stage, software failure data are not available. This paper presents the approach we took to deal with the above issues. The adaptive approach presented here continuously adjusts and evaluates the performance of the models as the software development proceeds. For the early-stage prediction, a simple and straightforward method is introduced which can be used when no failure data are available. This process, which is based on the adaptive approach and includes the early-stage prediction method, has been implemented in a software intensive development program in progress.

INTRODUCTION

As more and more failures attributed to software are observed, it is recognized that software reliability analysis is an inevitable task. However, although several software reliability models have been proposed [6], there is still no "standard" model. In reality, the needs of software reliability prediction force people to choose one (or more) models so that some software reliability numbers can be provided. The problem with this approach is that, at the beginning of a system development, there is no failure data available. Thus, no one knows which model best describes the software product. This approach is referred to as the blind approach.

Another approach is to apply various models and the results are compared with actual failure data at the end of the project. This way, the performance of different models can be evaluated [12]. The problem is, the software reliability can not be estimated until the very-late stage of the development, when software is almost ready to be delivered. This approach is referred to as the autopsy approach.

To cope with the above problems, we propose an approach that analyzes software reliability *adaptively*. That is, software reliability is modeled as the software development proceeds. First, we provide a rough estimation, to start the whole process. As the software is being developed, failure data become available, and software reliability can be predicted progressively. Comparing the actual failure data with the predicted numbers, we can see the trend of the software failure behavior, and determine which models are the most appropriate ones. When the software development reaches the final stage, modeling experience is also becoming more mature. The ultimate goal is to provide software reliability estimation using the model that best characterizes the failure behavior of the particular software product. Not only that, this process continuously provides estimation at each phase of the system development based on the most current failure information.

Note that even at the beginning of software development where failure data are not available, some assurance that the design is meeting its requirements is desirable. Therefore, a method that can provide a reasonable estimation before any actual failure data available is a benefit to the program. In a

[*] Contact author. Email: mlyin@west.raytheon.com. Tel: 714-446-4269. Fax: 714-446-3137.

survey provided in [6], three models have been identified as the "early-phase' models, i.e., Gaffney and Davis' phase-based model [3], Agresti and Evanco's Ada software defects model [1], and the Air Force's Rome Lab model [9].

The basic philosophy of these early-phase models is to do a prediction *based on as much information as possible*. For example, the phase-based model requires the information of discovered faults found during the design and implementation phases [3]; the Rome Lab's model considers a very comprehensive list of factors [9]. The Ada software defects model requires 4 product and 2 process characteristics [1]. Although detailed information is desirable, they are not necessarily available, or they may be very costly to obtain at the early stage of the program. In this paper, we propose a cost-efficient method, called the *early-stage prediction*, to be added to the adaptive prediction process for software reliability.

This adaptive process with the early-stage prediction method has been implemented in a software development program in progress. As more experience is gained and more failure data are collected, the performance of the early-phase prediction method is improved.

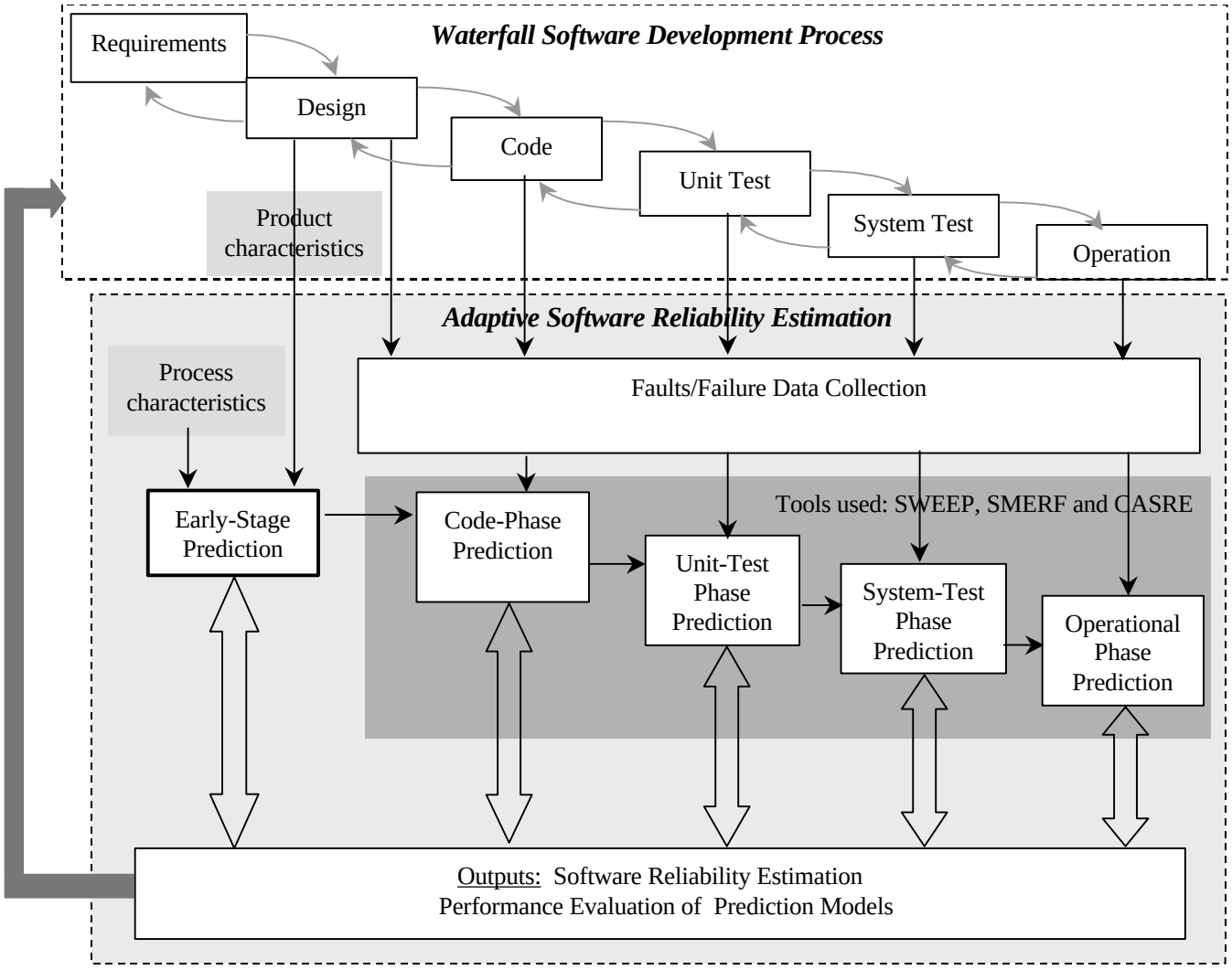

Figure 1. The Adaptive Software Reliability Prediction Process

THE ADAPTIVE APPROACH

The Process

The adaptive approach is integrated into the software development process, as shown in Figure 1. The

waterfall-software-development process [6] is used as the basis. As shown in the figure, a software product starts with some set of requirements, followed by *design, code, unit test, system test,* and the *operation* phases.

Five prediction activities are identified, i.e., early-stage prediction, code-phase prediction, unit-test-phase prediction, system-test-phase prediction, and finally the operational phase prediction. The early-stage prediction will be described in detail later. Once the software has been designed and implemented, information about discovered faults can be obtained, and code-phase estimation can be performed. The unit-test and system-test phase predictions can be conducted once those test data are available. When the software reaches the field (operational phase), software reliability growth is projected over its future use[1]. As failure data are being collected, the performance of the models can be evaluated. The outputs are not only the predicted software reliability number, but also an evaluation of the models. As illustrated in Figure 1, the outputs are fed back into the estimation process so that the software reliability models can be refined and justified. Moreover, these outputs are fed back into the development process to improve the product.

Tools Consideration

When faults/failure data are available, tools such as SWEEP (SoftWare Error Estimation Program), SMERF (Statistical Modeling and Estimation of Reliability Functions) and CASRE (Computer-Aided Software Reliability Estimation) can be applied. In particular, our process uses CASRE for the operational phase prediction and SMERF for the code-phase, unit-test-phase and system-test-phase prediction. SMERF and CASRE utilize the same set of models. SMERF is developed at the Naval Surface Warfare Center (NSWC) [6], and CASRE is developed in 1993 at Jet Propulsion Lab[10]. Eleven models are supported, i.e., geometric model, Jelinski/Moranda De-Eutrophication model, Littlewood and Verrall's Bayesian model, John Musa's basic execution time model, John Musa's logarithmic poisson model, Non-homogeneous Poisson (execution time), Brooks and Motley's discrete model, generalized Poisson model, Non-homogeneous Poisson (interval data), Scheiderwind's Max Likelihood model, and Yamada's S-shaped growth mode [6].

SWEEP is an implementation of the phase-based model [3]. It makes use of fault statistics obtained during the technical review of requirements, design, and the coding to predict the reliability during test and operation. Thus, SWEEP can be used before testing (after coding). On the other hand, CASRE and SMERF can be used in the system test phase. None of the above tools can be used for the very early-stage prediction where no fault or failure data are available. A methodology that provides estimation for this situation is the topic of the next section.

EARLY-STAGE PREDICTION

The purpose of this method is to provide a rough estimation on various software reliability measurements, based on the limited information. In particular, the only information required are the size of the software, measured by source lines of codes (SLOC), the maturity of the development process[2], and the schedule. Since only a rough estimation is expected, accuracy is not a main concern for the early-stage prediction. Instead, accuracy is the goal of the overall adaptive process, which will be achieved by continuously refining various models. The two basic assumptions are (1) the time between software failures is exponentially distributed (2) the occurrence of a failure is followed by the removal of the corresponding fault[3].

There are many research efforts devoted to the topic of imperfect software debugging. In particular, the asymptotic properties of software failure rates have

[1] The issues of asymptotic properties of software reliability have been studied [11], and different methods have been proposed.

[2] The software development process level, such as the SEI(Software Engineering Institute) CMM(Capability Meturity Model) or the ISO 9000 series of standard by the International Organization for Standardization, have been proposed to assist the assessment of inherent faults [5].

[3] This implies that we assume there is a one-to-one mapping between the faults and failures.

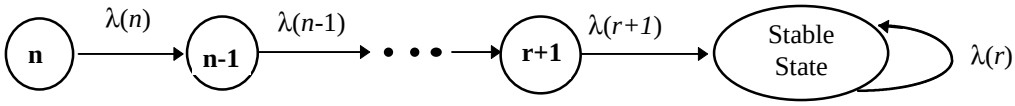

Figure 2. Software Failures Behavior Model for Early-Stage Prediction

been studied [11]. There is always a possibility that new faults will be introduced when removing a software bug. However, from a statistical point of view, the number of newly introduced faults is less significant when the total number of remaining faults is (relatively) large. It is only when the software product is reaching the mature stage, where the number of remaining faults and the number of introduced faults are in the same order of magnitude, should the imperfect debugging be concerned. This phenomenon is captured in our model as the "stable" state. Figure 2 shows the model that describes the behavior of software failures.

In this model, a software program is estimated to have *n* inherent faults at the beginning of the estimation. An assumption is made that the corresponding fault is removed when a software failure occurs. This will bring the software to the next state where the number of faults is decreased (one at a time). This process continues until the software reaches the stable state. In the stable state, the asymptotic failure rate phenomenon is observed. A failure rate function $l(i)$ is used in this model. This failure rate function $l(i)$ can be described in many different ways, according to the software failure behavior. For example, it can be described as a linear increasing function that is in proportion to the number of remaining faults, i.e., $l(i)=il$. Or, the failure rates can be described as a logarithmic increasing function, i.e., $l(i)=ln[i]´l$. This failure rate function should be a function of λ. The value of λ is then calculated based on the model, the parameters, and the failure rate function specified.

The key of this method is to find out the value of λ, using the information of the size of the code, the software process maturity level, and the duration *T*. *T* is the duration from the beginning time the software is measured (t_0) to the time the software is 'stable' (t_d). Theoretically, the selection of t_0 can be any time, for example, the time the software is finished compilation or the beginning of various phases indicated in the waterfall process.

According to [2] and [5], the actual failure data from different programs show that the stable time is approximately 4 years after delivery for a new program release. The stabilization period might be reduced to two years for subsequent program releases.

Once the starting time and the "stable" time are determined[4], the next step is to estimate the number of inherent faults, denoted as *n*, and the number of remaining faults, denoted as *r*.

Estimating the numbers of inherent faults and remaining faults

In order to solve the model described in Figure 2, the number of inherent faults, i.e., *n*, and the number of remaining faults, i.e., r, need to be determined. The inherent faults are the faults existing at time t_0; remaining faults are the faults existing at time t_d. A wide-used method to determine the number of inherent faults is through the use of fault density[5].

There are several studies on estimating the fault density. Musa's survey [7][8] provides fault density estimated for different software life-cycle phases. As presented in [8], the mean inherent fault density remaining at the beginning of different phases is estimated based on actual failure data from many different programs. As an example, the inherent fault densities for different phases are summarized in the following table.

Table 1

Phase	Faults/KSLOC
Coding (after compilation/assembly)	99.5
Unit Test	19.7
System Test	6.01
Operation	1.48

(copied from [8], Table 5.4)

[4] Note that this is only a rough estimation.

[5] Although Hatton [4] disagrees with this approach, the size of the code times the fault density is commonly used in the field.

The work of Aagresti and Evanco's Ada software defects estimation method [1] recognizes the differences in the way organizations develop software for software reliability prediction. Both *process* characteristics and *product* characteristics are considered in the overall software defects model. Moreover, Keene [5] proposed an approach that applies the software process levels and the size of the code to predict the number of inherent faults. As an *example*, the following table shows the relationship of the inherent fault densities at the beginning of the operational phase and the software process levels.

Table 2

SEI CMM Level	Faults/KSLOC
5	0.5
4	1.0
3	2.0
2	3.0
1	5.0
Un-rated	6.0

For the value of *r*, i.e., the number of remaining faults, the observation in [2] and [5] suggests that, after four years of deployment, the number of software faults be reduced to a level less than 10% of the level at deployment. Thus, one way to conservatively estimate the value of *r* is to use 10% of the fault density estimated at the beginning of the operation phase.

Calculating l

Define a sequence of non-negative real-valued infinite random variables X_0, X_1, X_2, … , X_i, …. Each of these random variables represents the time between two consecutive failures. Recall that exponential distribution has been assumed. The value of λ is assessed by utilizing the relation that $E[X_1 + X_2 + \ldots + X_{n-r+1}] = E[X_1] + E[X_2] + \ldots + E[X_{n-r+1}]$. In other words, we have the equation $1/\lambda(n) + 1/\lambda(n-1) + \ldots + 1/\lambda(r+1) = T$[6]. Given the failure rate function $l(i)$ and the values of *n, r* and *T*, the value of λ can be calculated.

Stable State MTBF

Plugging λ into the failure rate function with parameter *r*, i.e., $\lambda(r)$, the stable state MTBF can be estimated, i.e., $1/\lambda(r)$.

[6] For exponential distribution $1-e^{-\lambda t}$, the expected value is $1/\lambda$.

Expected Number of Failures Occurred

The expected number of software failures occurred by time *t* is calculated in the following way. First, we estimate the state the software is expected to be in at time *t*. This can be done by calculating $\Sigma(i \cdot P_i)$, where *i* is the number of existing faults (the state number), and P_i is the probability that the software is in state *i* at time *t*. Denote the expected number of existing faults at time *t* as *k*. Then, the expected number of software failures that have occurred by time *t* is *n-k*.

Software MTBF prior to the Stable State

Suppose at time *t*, the number of existing faults is predicted (by the above method) to be *k*, then the MTBF at time *t* can be estimated as $1/\lambda(k)$. This can be interpreted as the mea time between software failures if no further faults are removed.

Software Reliability

According to the standard definition of reliability[7], the software program's reliability is the probability that at time *t*, the software is still in state *n* (no failure has occurred yet). This reliability number depends on the value of *l*, which in turn is dependent on the other parameters and the failure rate function specified.

Improvement of the Method

Although this method only provides a rough estimation, the actual data collected in the later phases will give us feedback on the method and the parameters used. Furthermore, the experience from actually implementing the process will improve the overall approach, which can be used for other programs. For example, the fault density level at the stable state used now is 10% of the level when the software is deployed. This number can be refined or justified, as more experience is gained. The accumulation of this experience over time can be added to the confidence in the reliability parameters, which can then be used in upcoming programs.

In the next section, an example is used to demonstrate the early-stage prediction, and the feedback gained from later phases prediction activities.

[7] The reliability is defined as the probability that the component operates correctly throughout the interval $[t_0, t]$ given that it was operating correctly at time t_0.

AN EXAMPLE

In this example, we consider a software product whose size is 360 KSLOC (K Source Lines Of Code). The software process applied is rated as SEI CMM Level 4. The duration *T* is assumed to be 4 years. Based on this information, the early-stage prediction method suggests the fault density at the beginning of the operation phase is 1.0 per thousand lines of code, i.e., 360 faults, if the process-driven fault density model is applied (Table 2). If Table 1 is used, then the fault density at the beginning of operation phase is 1.48, i.e., 533 faults. These two models give us a rough estimation on the number of inherent faults at the beginning of the operation phase. According to the discussion presented in the previous section, we can derive various software reliability measurements based on this information.

As the software development is progressing, faults/failures data are collected. The tool SWEEP was used to perform a phase-based model prediction [3]. Eight phases were specified, e.g., preliminary design, detailed design, code, unit test, integration test, final test, system test, and operation phases. Figure 3 through Figure 7 show the adaptive predicted fault density for each phase based on different sets of available failure data. Specifically, Figure 3 is the prediction made at the beginning of code phase, when only the defects found in preliminary design and detailed design are known. Figure 4 is the prediction made at the beginning of unit test, and so on.

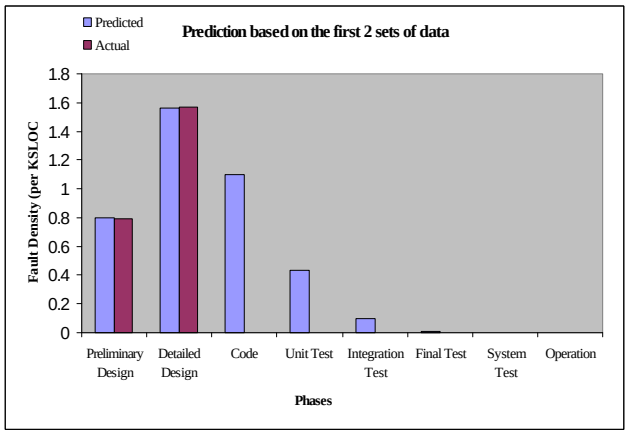

Figure 3. SWEEP Prediction based on 2 sets of data

The stage of the project is currently at the beginning of system test. Therefore, only the failure data up to the final-test phase are available. The predictions show that the operation phase fault density based on the most updated failure data, i.e., 0.99, is very close to our early-stage predictions (1.0 if using process-driven model, 1.48 if using Musa's survey). This example demonstrates that we can earn more confidence in the model that we chose at the earlier stage, by the predictions performed at the later phases.

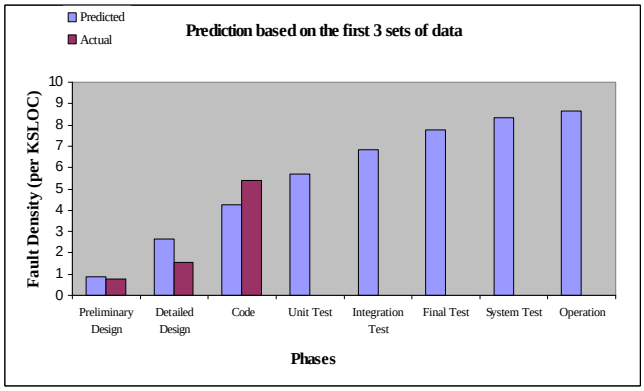

Figure 4. SWEEP Prediction based on 3 sets of data

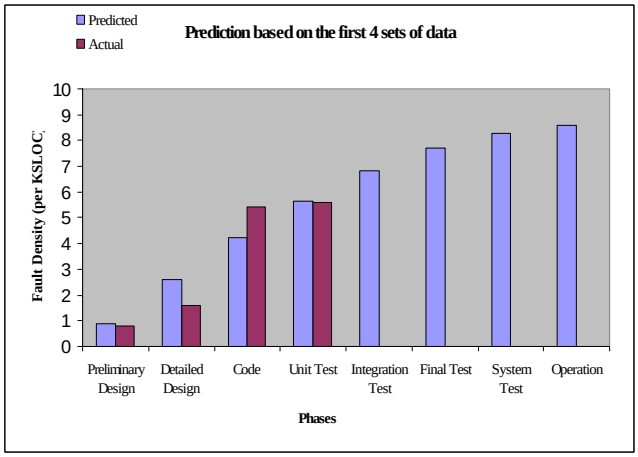

Figure 5. SWEEP Prediction based on 4 sets of data

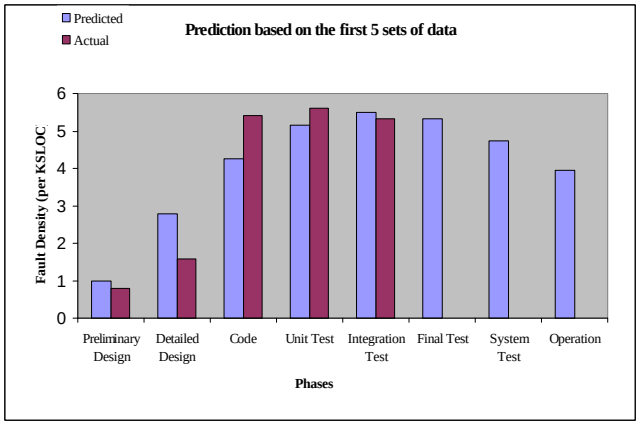

Figure 6. SWEEP Prediction based on 5 sets of data

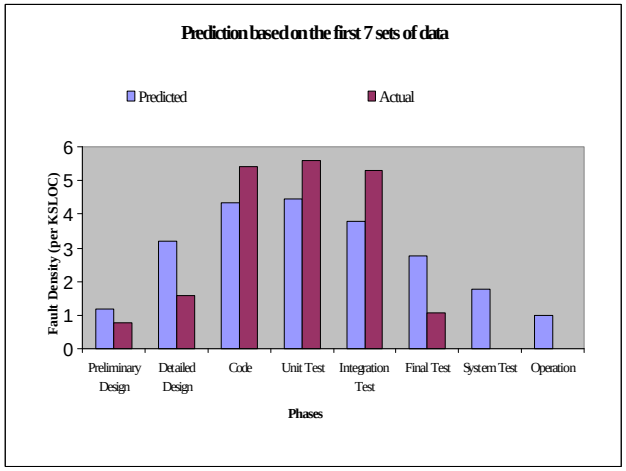

Figure 7. SWEEP Prediction based on 6 sets of data

CONCLUSION

We have presented an adaptive approach, which is integrated with the software development process, to estimate the software failure behavior. This approach has been implemented in an ongoing software development program. The key feature of this method is that the prediction is improving as the software proceeds. Our basic philosophy is that, since the software product is evolving continuously, the software reliability prediction should be improving continuously.

Moreover, a method that can assess software reliability in the early stage is presented. This method requires only very limited information about the software product and the process. The asymptotic property of software failure rates is recognized in the model. While most early-phase software reliability prediction methods focus on how to provide a precise prediction with the limited information, we provide a rough estimation as a starting point of the overall prediction process. The accuracy of the estimation is the goal of the overall process. The approach presented here is readily performed and should provide adequate initial software reliability estimation. As more experience in this early-stage prediction is gained, the method can be improved and benefit other software development products.

REFERENCES

[1] W.W. Agreti, and W.M. Evanco, "Projecting Software Defects From Analyzing Ada Design," IEEE Transactions on Software Engineering, Vol.18, No.11, Nov.1992, page 988-997.

[2] Ram Chillarege, Shriram Biyani, Jeanette Rosenthal, "Measurement of Failure Rate in Widely Distributed Software," *Fault Tolerant Computing Symposium (FTCS)*, 1995, page 424-433.

[3] J.E. Gaffney and Davis, C.F., "An Automated Model for Software Early Error Prediction (SWEEP)," Proceedings of the 13th Minnowbrook Workshop on Software Reliability, July 1990.

[4] L.Hatton, "Reexamining the Fault Density – Component Size Connection," IEEE Software, March 1997, pp. 89-97.

[5] S.J. Keene, "Modeling Software R&M Characteristics," ASQC Reliability Review, Part I and II, Vol 17, No.2&3, 1997 June, pp.13-22.

[6] Michael R. Lyu (editor), *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

[7] John Musa, "A Theory of Software Reliability and Its Application," IEEE Transactions on Software Engineering, Vol. SE-1, No.3, Sep. 1975, page 312-327.

[8] John D. Musa, Anthony Iannino, Kazuhira Okumoto, *Software Reliability - Professional Edition*, McGraw-Hill, 1990.

[9] Rome Laboratory (RL), Methodology for Software Reliability Prediction and Assessment, Technical Report RL-TR-92-52, volumes 1 and 2, 1992.

[10] A.P. Nikora, "CASRE User's Guide," Jet Proopulsion Laboratories, August 1993.

[11] M.C.J. Van Pul, *Statistical Analysis of Software Reliability Models*, Stichting Mathematisch Centrum, Amsterdam, 1993.

[12] A. Wood, "Predicting Software Reliability," IEEE Computer, Nov. 1996, pp. 69-77.